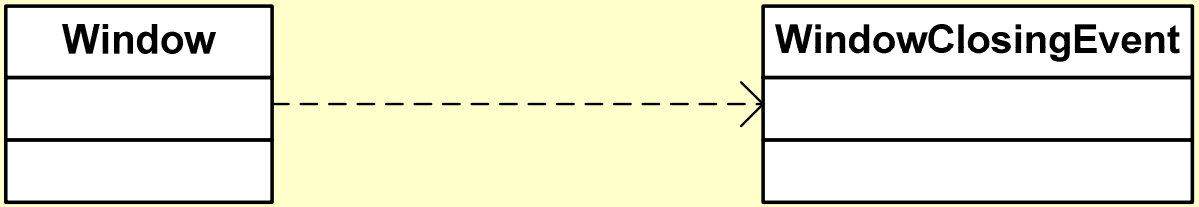
Object-Oriented Design

UML 2.0: Relationships

Relationships

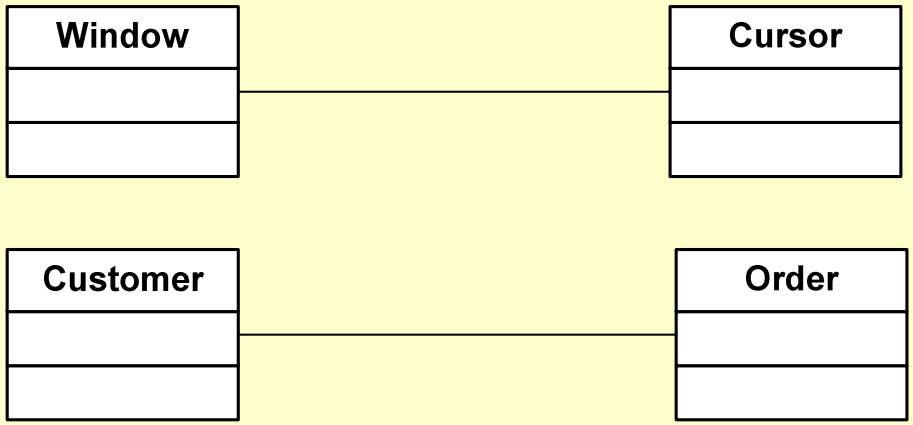
- UML 2.0 defines some concepts to model interactions between classes, each corresponding to a graphical element in the class diagram.
- From the weakest relationship to the strongest:
  - Dependency
  - Association
  - Aggregation
  - Composition
  - Generalization

Relationships: Dependencies



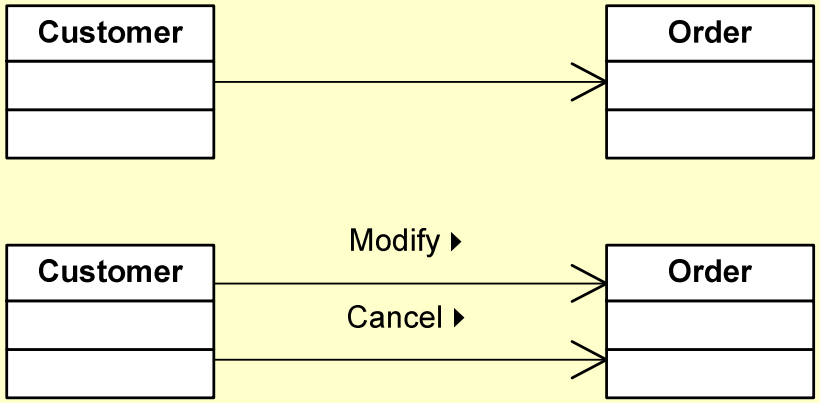
- Dependency is the weakest relationship between classes
- Class A depends on class B if A uses B in a way or the other
- Usually, it is a temporary interaction, of which no trace is kept after use
- If you can say "A uses B", chances are that there is a dependency relationship between A and B

Relationships: Associations



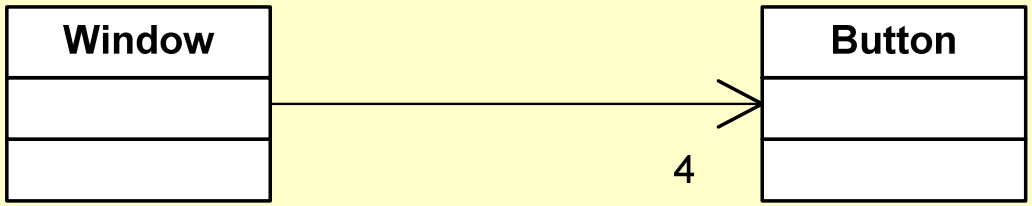
- A stronger kind of relationship
- Class A is associated to class B if A is connected to B for a certain amount of type
- Class A and B have an independent life
- If you can say that "A has a B", chances are that there is an association between A and B

Associations: Navigability & Names



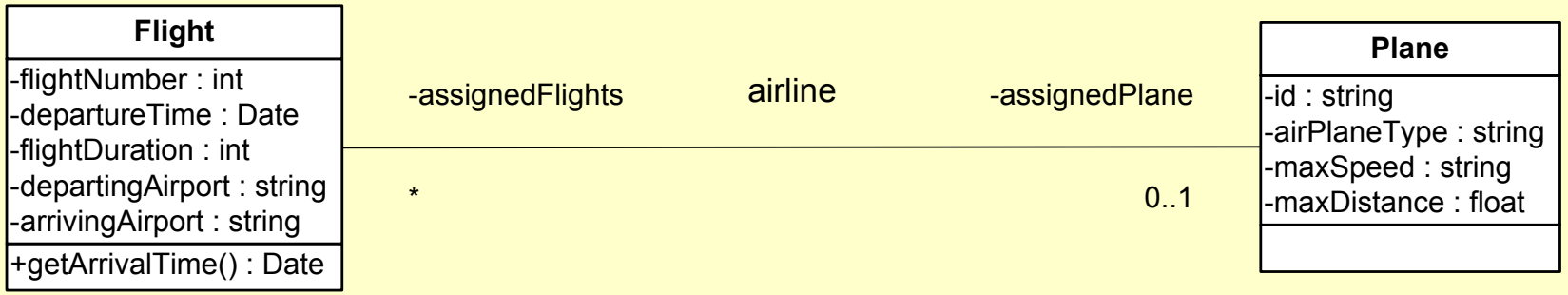
- Arrows indicate the possibility to navigate from A to B
- Navigability from class A to class B means that from an instance of class A we can access the associated instance of class B
- When the association can be navigated in both directions, no arrow is drawn
- The association can be named
  - optionally, when there is only one relationship
  - mandatorily, to distinguish two relationships between the same classes

Associations: Multiplicity



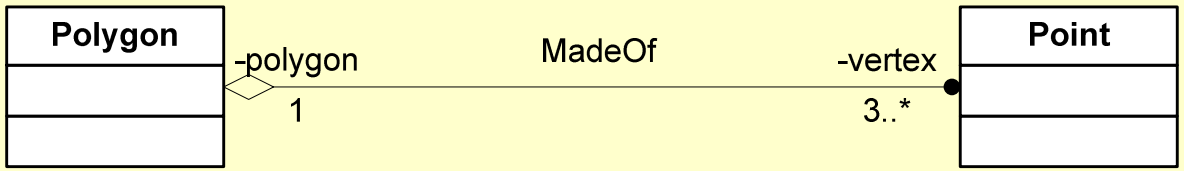
- Usually, association models permanent relationships
- Therefore, they are often used to represent class attributes
- Multiplicity indicates how many instances of a class are involved in the association
- Default multiplicity is 1

Associations: Roles



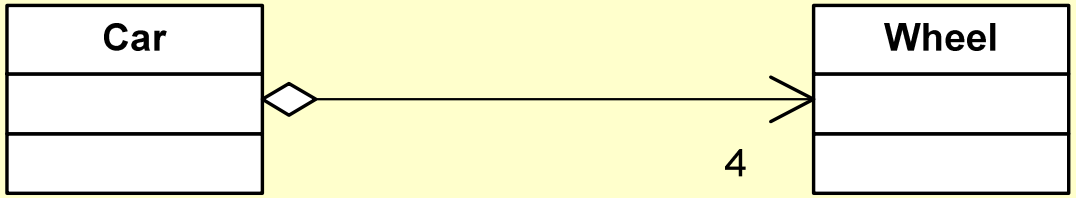
- assignedFligts and assignedPlane are the roles of the relation (or association) airline
- assignedFlights and assignedPlane are attributes by relation

Associations: Ends' Ownership



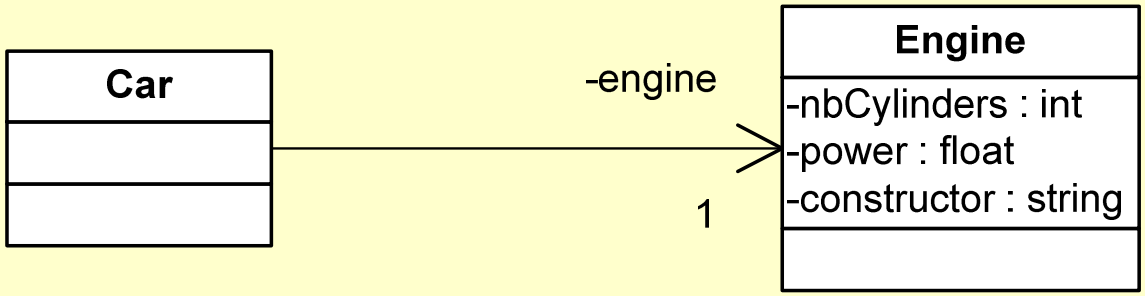
- A dot on an association's end means that the end belongs to the class on the opposite end
- No dot means that the end belongs to the association
- In the example :
  - `vertex` is an attribute owned by class Polygon
  - `polygon` is an attribute owned by relationship `MadeOf`

Relationships: Aggregation



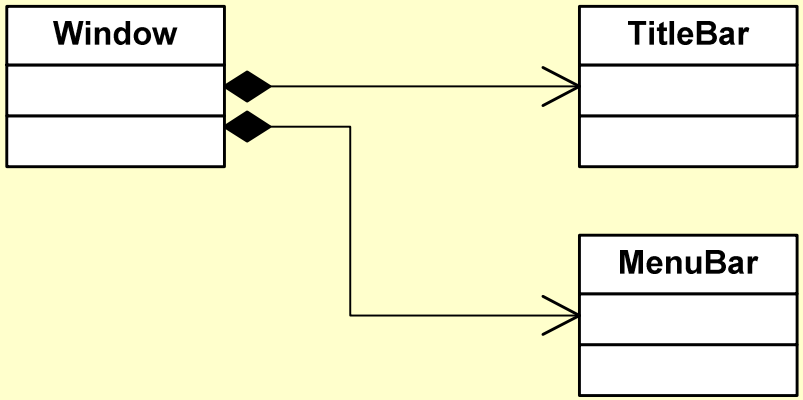
- Aggregation is a stronger relationship than association
- Describes a relationship of property between classes
- To be used when you can say "A owns a B"

Attributes by relation



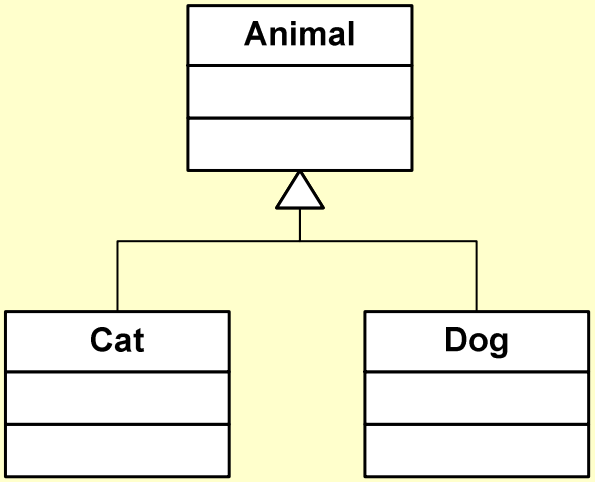
- engine is an attribute of Car and is called an attribute by relation
- If the name of the role is the same as the name of the class, usually it is omitted

Relationships: Composition



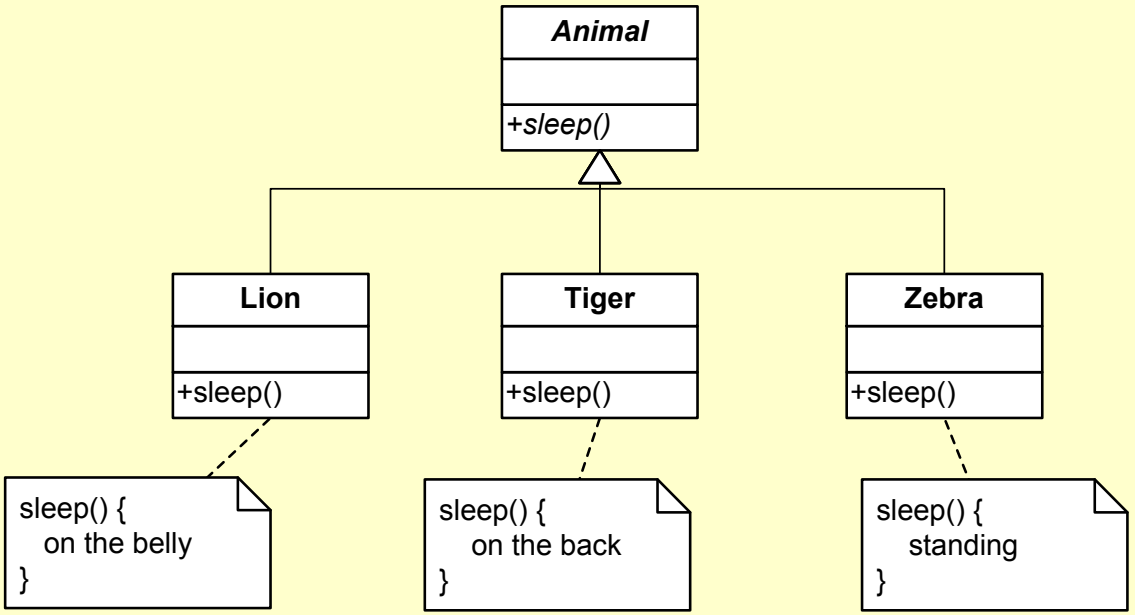
- Composition is the strongest relationship between classes
- Indicates total possession of class B by class A
- At every single time, the owned class can be into only one composition relationship
- The owned class cannot exist before the container class
- If proprietary class A is destroyed, so are all classes which are connected to A by composition
- To be used when you can say "B is part of A"

Relationships: Generalization



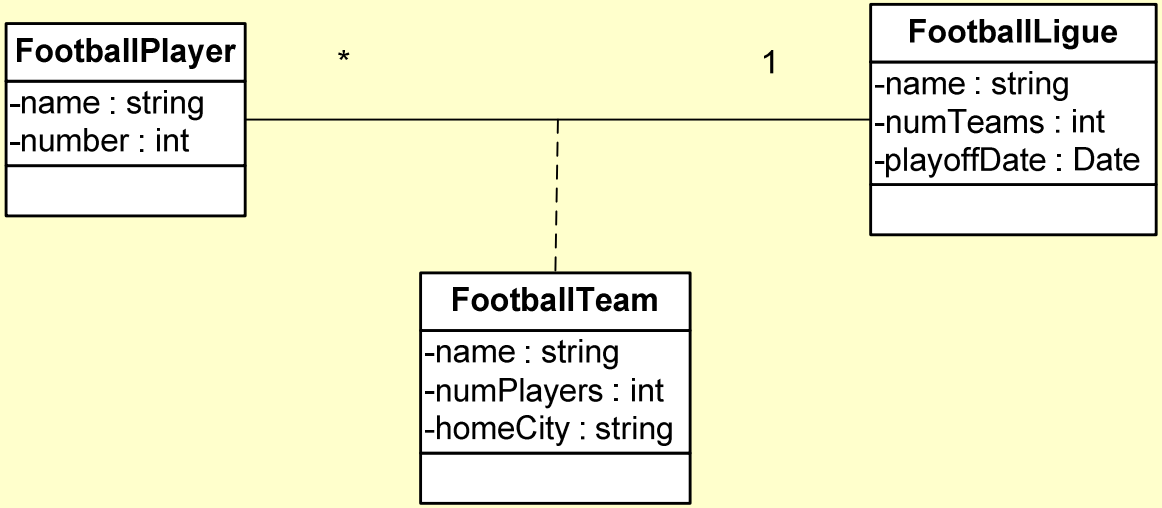
- Generalization or Inheritance relationships indicate that a class B is a specialization of class A
- To be used when you can say "B is a A"
- Usually they have neither name, nor multiplicity

Abstract Classes



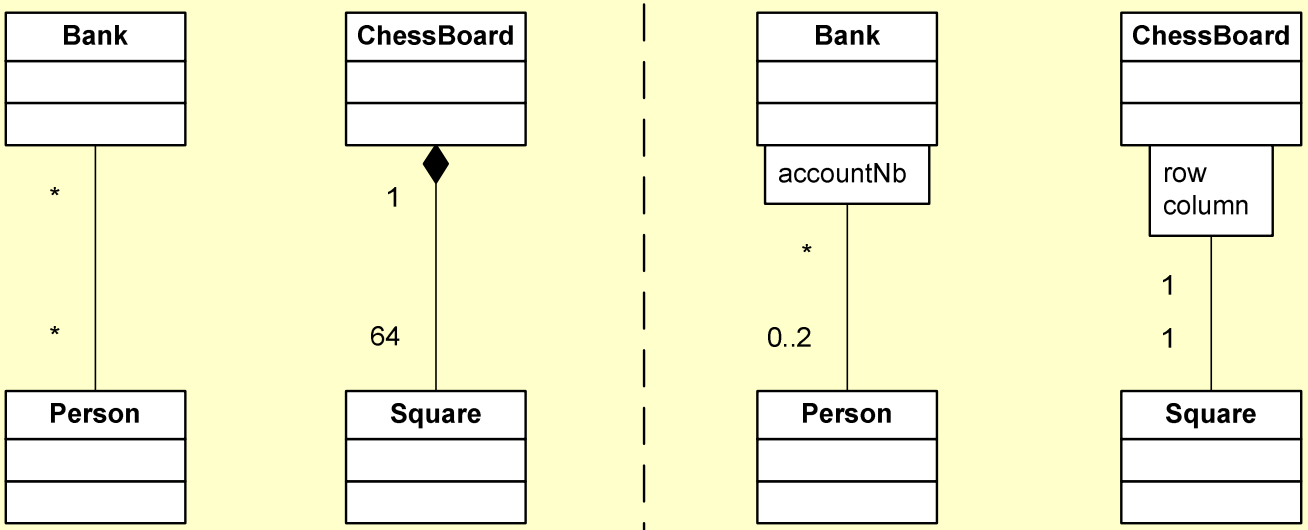
- An abstract class is a class which has at least one operation whose implementation is absent

Relationships: Association Classes



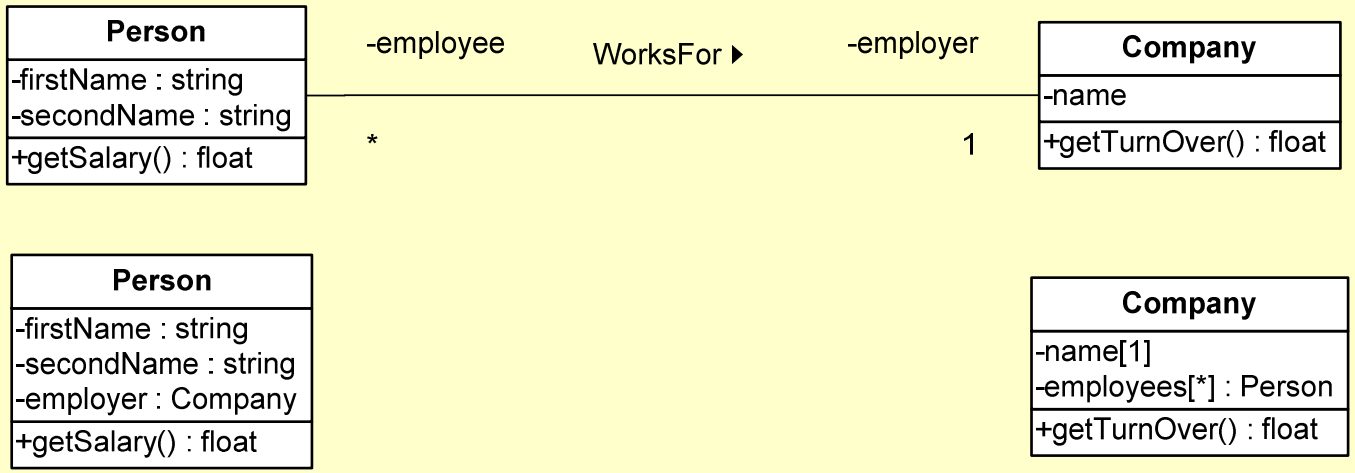
- Often the association between classes is not a simple structural connection
- When it's complex and carries a lot of information, an association class can be used
- An association class is an association with names and attributes

Association Qualifiers



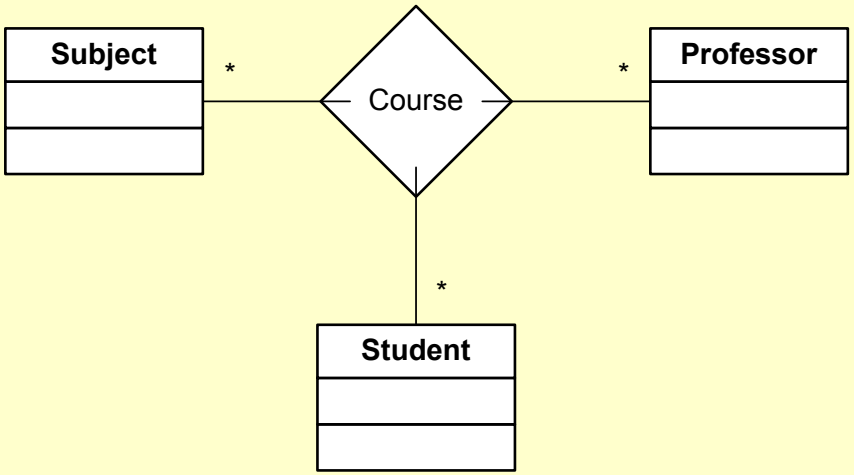
- Sometimes associations between two classes are *indexed* on a *key*
- Usually, the key is an attribute of the target class
- Association qualifiers are the UML 2.0 equivalent of association tables for programming languages (maps, hash tables, etc.)

Modeling an association

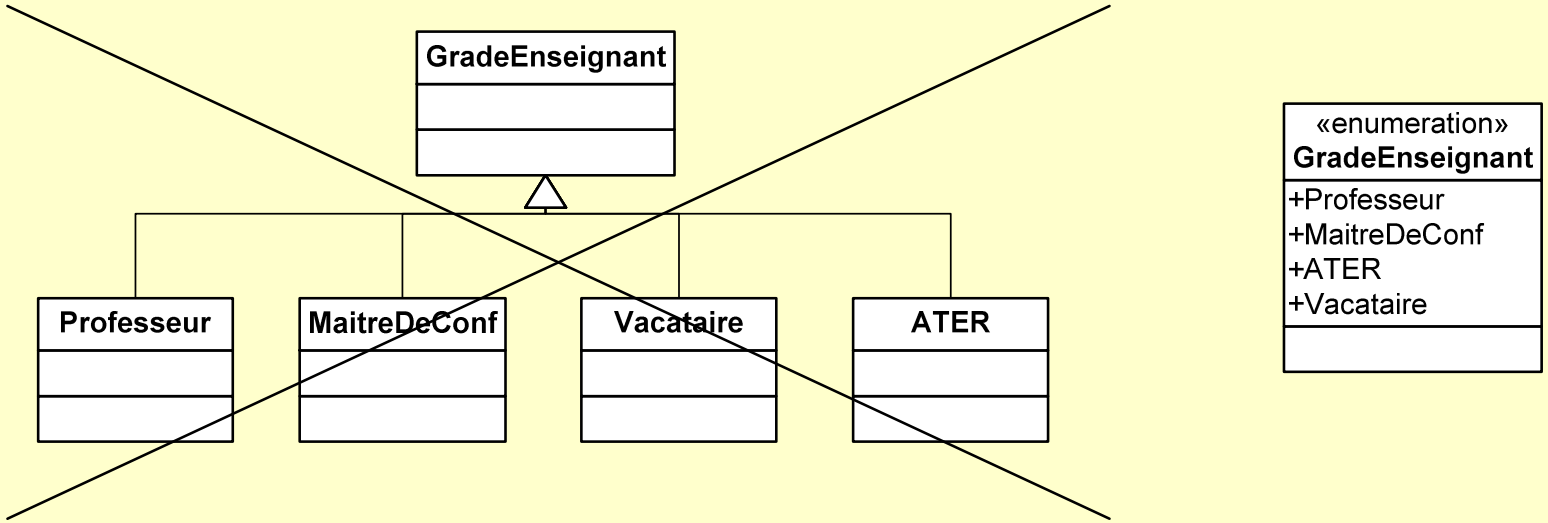


- Two alternative ways to model an association
- First one explicitly models the association
- Second one implicitly: it is the ways association are usually implemented

N-ary association



Enumerations



- Enumerations are classes that represent objects which can take only a finite number of values